Deep Learning - Practicum 5 - Deep Q Net

Topics covered: Deep Q Network, GYM environment Cartpole

Deliverables: Your submission for this practice includes two files, named `agent.py` and `model.py`. This practice uses Keras with Tensorflow, and GYM.

Objectives: Deep reinforcement learning combines artificial neural networks with a reinforcement learning architecture that enables software-defined agents to learn the best actions possible in virtual environment in order to attain their goals. In this practice, we're to train a neural network model that approximate the Q-value function and apply it to instruct an agent to play `cartpole` "game" (provided by GYM environment). After completing all the tasks, you'll:

- understand how reinforcement learning works using deep neural networks.
- implement the Deep Q-Network (DQN) on the game of Cartpole.

GYM Environment

Gym is a useful toolkit for a learner of reinforcement learning algorithm. It currently provides us different environments in several categories. In this practice, one of the classical environments – CartPole-V1 is used to test the implemented Deep Q-Network. We're going o train a DQN that play the game - moving the cart to prevent it from falling over.

Read the documentation for installation of `gym` package for python and the environment data (`state`, `reward` etc.) returned by the `gym` interface.

Ideally, agent would be trained on the image pixels from game frames. Considering its computational cost, our agent will be trained on the `state` return by `gym evn` each step, instead of the game image pixels,

The code below (in `cartPoleTest.py`) will run an instance of the environment. It is printing out the environmental data and rendering the environment at each step. You should see a window pop up rendering the classic cartpole game played by a random agent.

```
import gym

env = gym.make('CartPole-v1')
[next_state, reward, done, info] = env.reset()
print([next_state, reward, done, info])

done=False

while(not done):
      action = env.action_space.sample()
      next_state, reward, done, info=env.step(action)
      print(next_state, reward, done, info)
      env.render()

env.close()
```

Tasks

Download the starting code (`cartpoleDQN-start.zip`) from course moodle page. We'll start with the top-level file `main.py`. The training framework is given in this file. Give yourself enough time to read the code and comments in the function `train()`, try to associate it with what you've learned of DQN in the lesson.

Another given file is `momory.py` that contain the definition of class `Memory`. The class is the implementation of memory pool.

Your tasks is to complete and submit two files `agent.py` file and `model.py`. The code for your start in `model_satart.py` and `agent_start.py` respectively. **DO NOT** change any script in the starting code unless specified.

Task 1. Class Model

In this task you're to build the policy-net by defining a deep neural network in the `constructor` of class `Model` in `model.py` file. Considering the size of the state (4), a fully connected neural network is recommended.

```
# Deep Model Used by the Agent
class Model:
    def __init__(self, state_size, action_size):
        # state_size -- input, action_size -- output
        self.state_size = state_size
        self.action_size = action_size

          # Build a neural network model and assign it to the variable
self.model
          # e.g.  self.model = Sequential()
          # cause the size of state is just 4, a convnet seems not a good choice
          # fully connected multi-layered feed forward neural network can be
considered
          # # # Your code goes here


    def _save_model(self):
        self.model.save_weights("weights_cartpoledqn.h5")

    def _load_model(self):
        self.model.load_weights("weights_cartpoledqn.h5")
```

Task 2 Class Agent

The constructor of class `Agent` is defined below. Notice that the memory pool `mempool` and policy net `net` are members of `Agent`.

```
class Agent:

    def __init__(self, state_size, action_size):
        self.state_size = state_size
        self.action_size = action_size

        # create memory pool
        self.mempool = memory.Memory()

        # create neural network model, we call it policy net
```

```
        self.net = model.Model(state_size=state_size, action_size=action_size)

        # parameters for q-learning
        self.gamma = 0.95    # discount rate
        self.epsilon = 1.0   # exploration rate
        self.epsilon_min = 0.01       # the low bound of espison
        self.epsilon_decay = 0.995    # used for adjusting the epsion along with
the time/steps
```

Task 2.1 get action

Implement the `get-action` method for class `agent` in file `agent.py`. The method uses policy net `self.net` to decide the action to take.

```
def get_action(self, state):

        # get action just using policy net
        # return the action
        # # # Your code goes here
```

Task 2.2 epsilon greedy policy

Implement the `eps-greedy` method for class `agent` in file `agent.py`. The method is also used to decide the action to take. But it applies epsilon greedy policy, it returns either a random action or the action predicted by the policy net.

```
def eps_greedy(self, state):

    # get action using epsilon greedy policy together with policy net
    # self.epsilon can be used for the greedy policy
    # return the action
    # # # Your code goes here
```

Task 2.3 train policy net

Complete the method `train_ploicy_net` to train your policy net.

```
def train_policy_net(self, batch_size):

    # Get minibatch from memory pool
    minibatch = self.mempool.sample_minibatch(batch_size)

    # initialize input and output
    states, targets = [], []

    # generate training data set for training the policy net
    for state, action, reward, next_state, done in minibatch:

        # compute the targets, for terminal(done) and non-terminal(not done)
        # # # Your code goes here

        # append the state and target to states(inputs) and targets(outputs)
        # # # Your code goes here

    # train the policy net
```

```
    # # # Your code goes here

    # Decay the value of epsilon
    if self.epsilon > self.epsilon_min:
        self.epsilon *= self.epsilon_decay
```

Task 3 Train and Test

Run the `Train` and `Test` in the file `main.py` to train the agent and let it play the cartpole game. It will take less than 15 minutes for running 200 episodes, maximal 200 steps each episode, and the highest score the agent could get is 200.